Lifting Micro-Update Models from RTL for Formal Security Analysis

Adwait Godbole, Kevin Cheang, Yatin A. Manerkar, Sanjit A. Seshia

La Jolla, San Diego, California – ASPLOS 2024

Challenge: Formally Verifying Software for Microarchitectural Vulnerabilities

```
cryptobox:
        addi sp,sp,-128
        sd ra,0(sp)
        ...

main:
        addi sp,sp,-36
        sd ra,0(sp)
        ...
        call cryptobox
```

Challenge: Formally Verifying Software for Microarchitectural Vulnerabilities

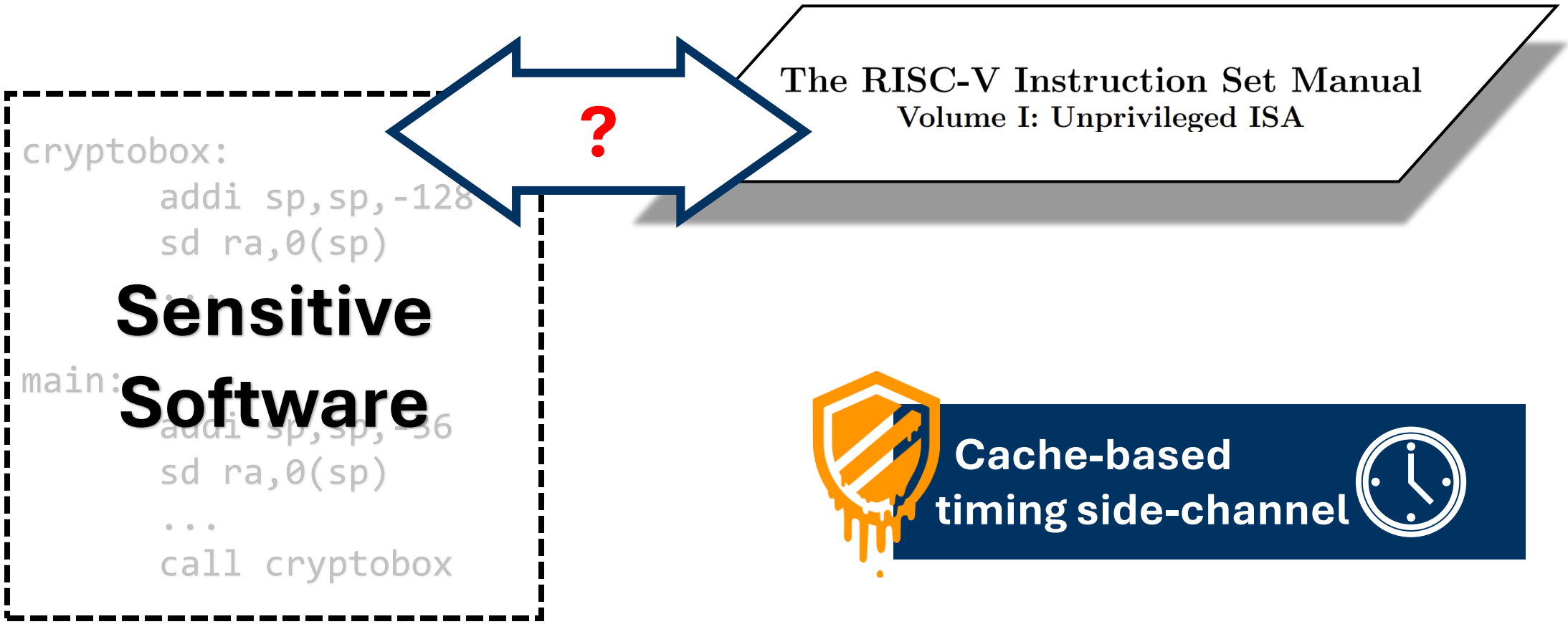
```
cryptobox:
        addi sp,sp,-128
        sd ra,0(sp)
        ...

main:
        addi sp,sp,-56
        sd ra,0(sp)
        ...
        call cryptobox
```

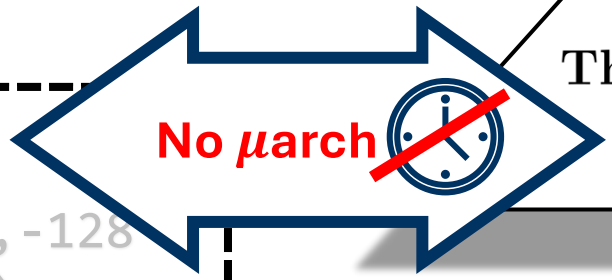
Sensitive Software

Cache-based timing side-channel

Challenge: Formally Verifying Software for Microarchitectural Vulnerabilities



Challenge: Formally Verifying Software for Microarchitectural Vulnerabilities



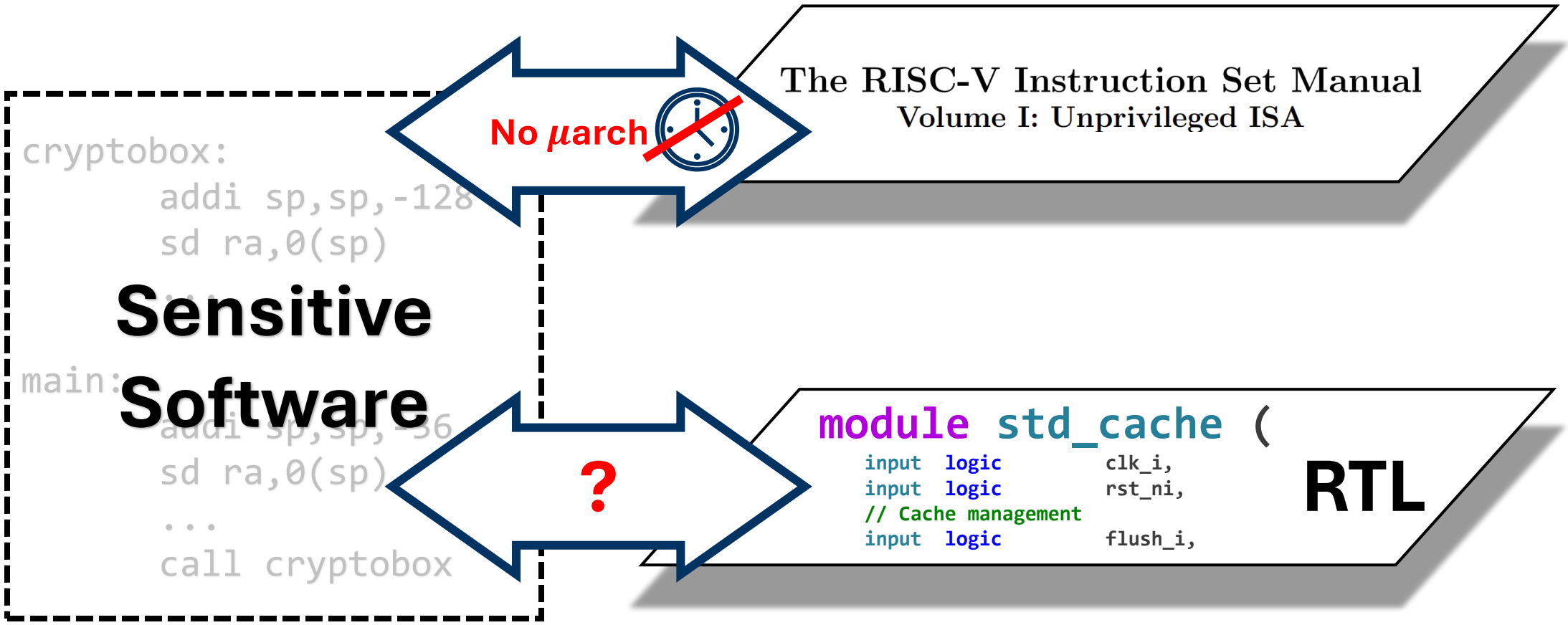
The RISC-V Instruction Set Manual
Volume I: Unprivileged ISA

```
cryptobox:
        addi sp,sp,-128
        sd ra,0(sp)
        ...
main:
        addi sp,sp,-36
        sd ra,0(sp)
        ...
        call cryptobox
```

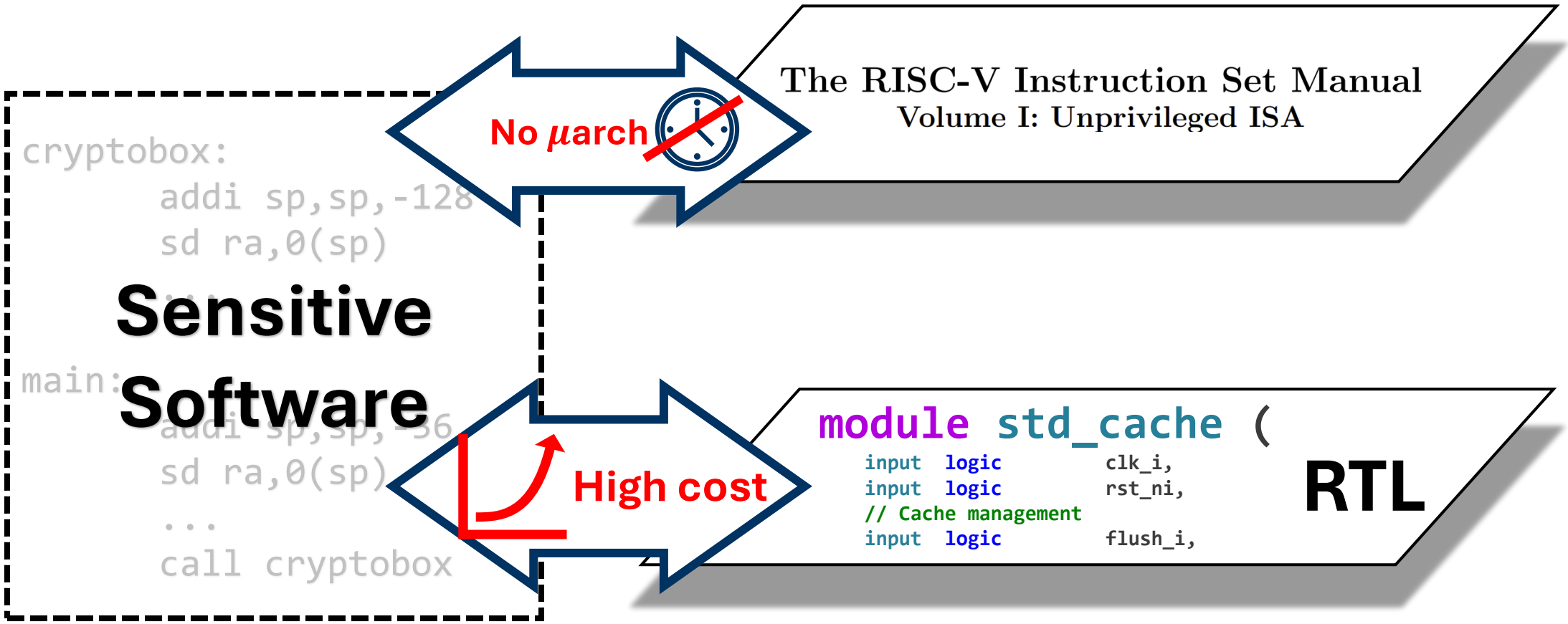
Sensitive Software

Cache-based timing side-channel

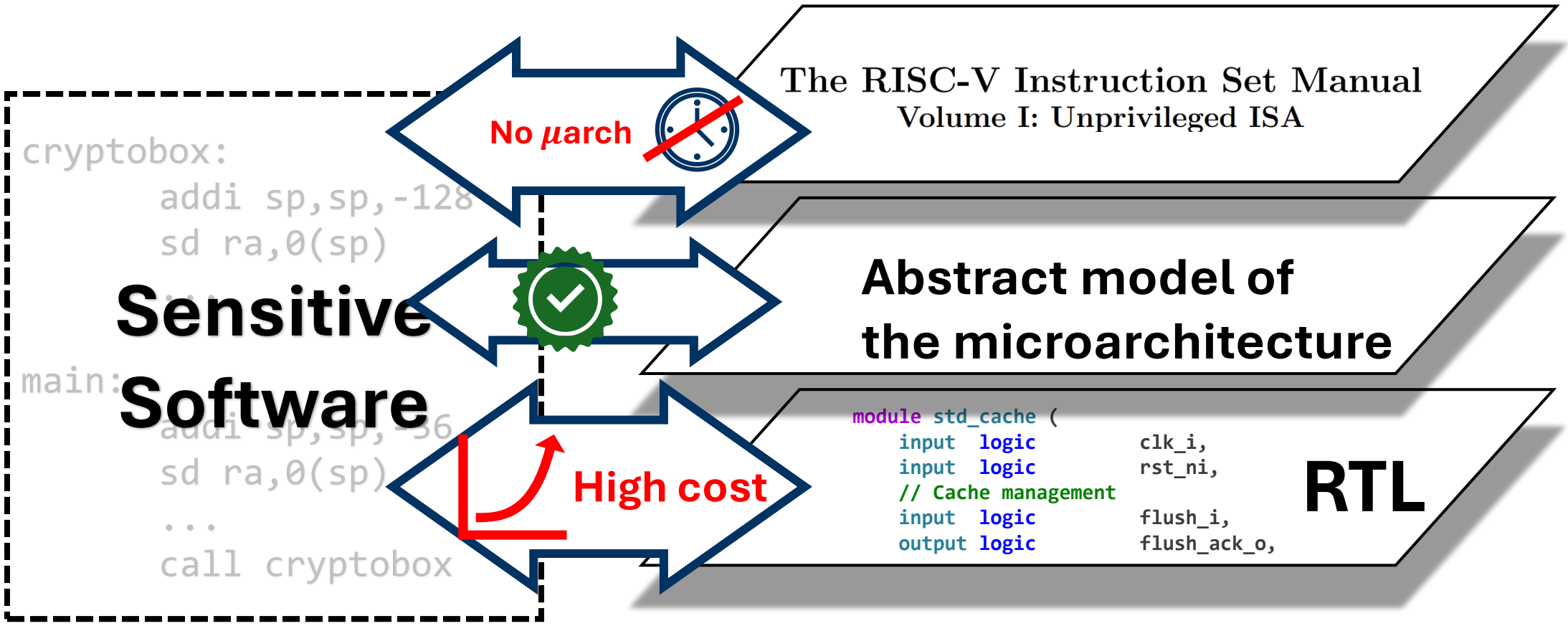
Challenge: Formally Verifying Software for Microarchitectural Vulnerabilities



Challenge: Formally Verifying Software for Microarchitectural Vulnerabilities



Abstract Microarchitectural Models



e.g., Cheang et. al. [CSF '19], Guarneri et al. [S&P '20], et. al.

Abstract Microarchitectural Models: where do they come from?

Manually developing models is tedious and error-prone!

Abstract Microarchitectural Models: where do they come from?

Manually developing models is tedious and error-prone!

Can we automate this?

Our Contribution: Automated Lifting

Micro-update model

Abstract modeling framework to capture a design-slice

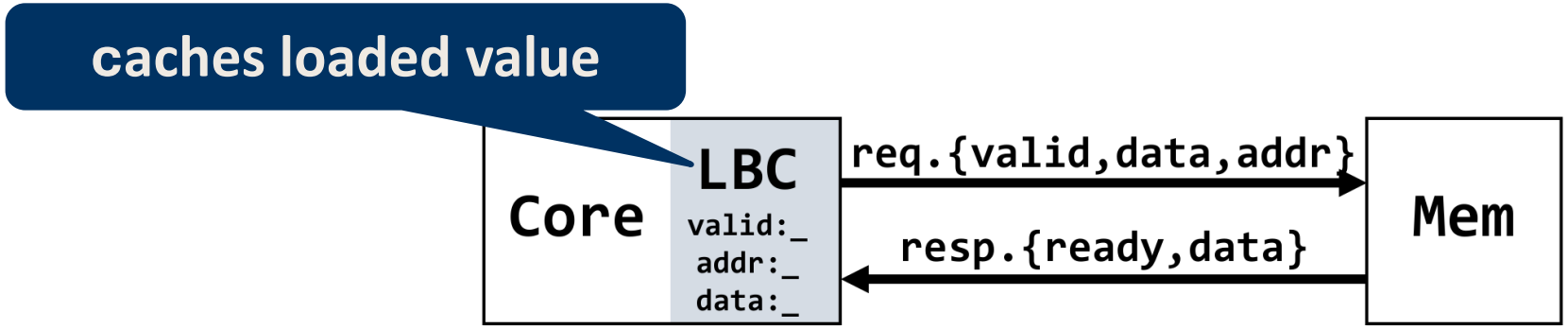
Model lifting technique

Automatically generate micro-update models from RTL

Outline

- **Motivating Example**
- Our formalism: the micro-update model
- Problem Statement
- Approach Highlights
- Evaluation

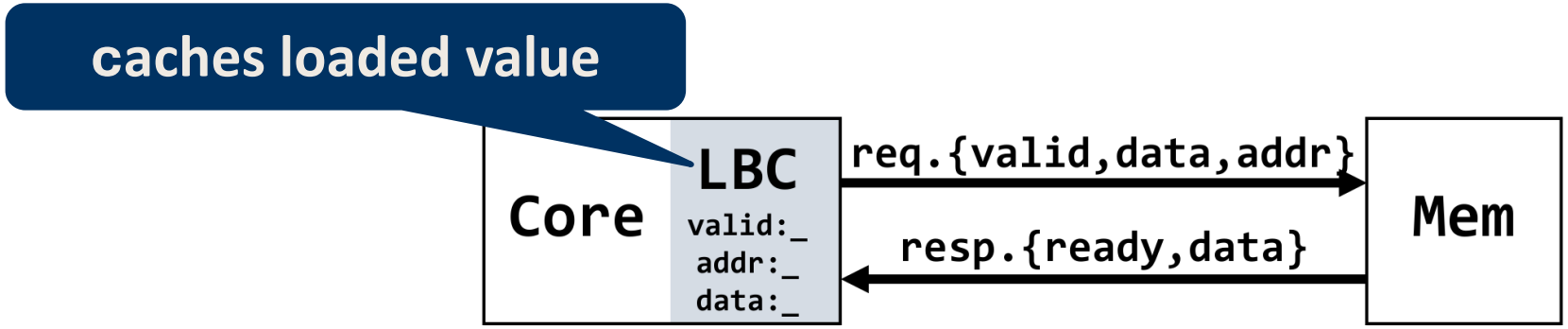
Example: Load Buffer Cache (LBC)



If address matches with cached value: replay value

If address does not match: new memory request

Example: Load Buffer Cache (LBC)

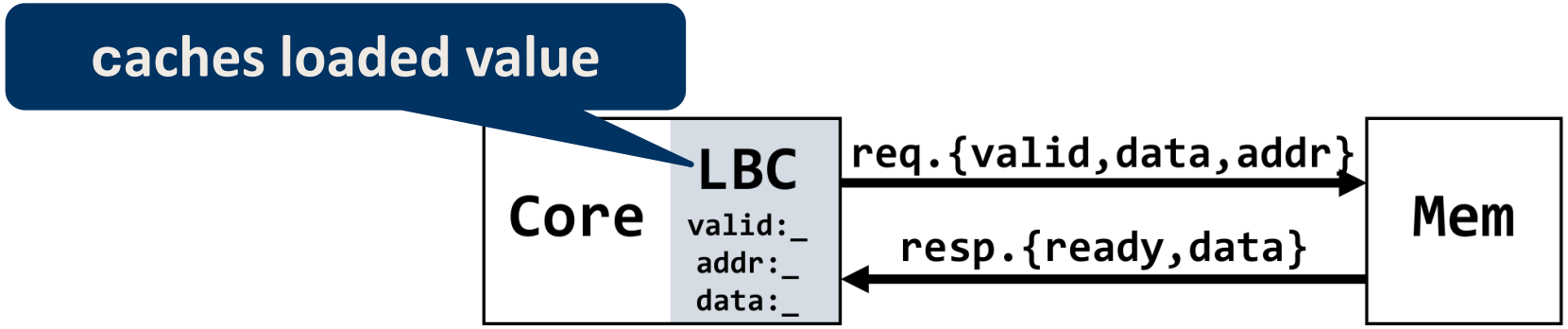


If address matches with cached value: **replay value**

timing side-channel

If address does not match: **new memory request**

Example: Load Buffer Cache (LBC)



If address matches with cached value: **replay value**

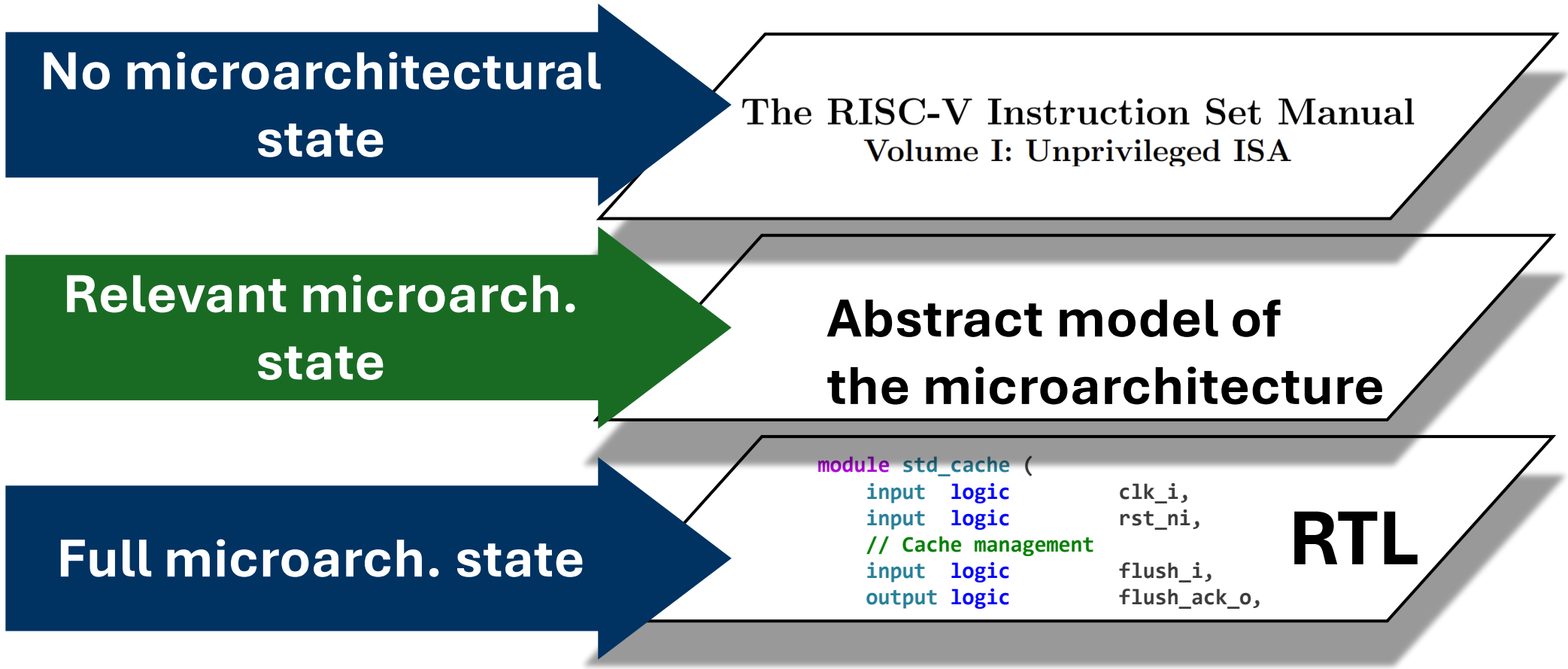
If address does not match: **new memory request**

Security analysis must take into account behaviour of the LBC.

Outline

- Motivating Example
- **Our formalism: the micro-update model**
- Micro-update model synthesis problem
- Approach Highlights
- Evaluation

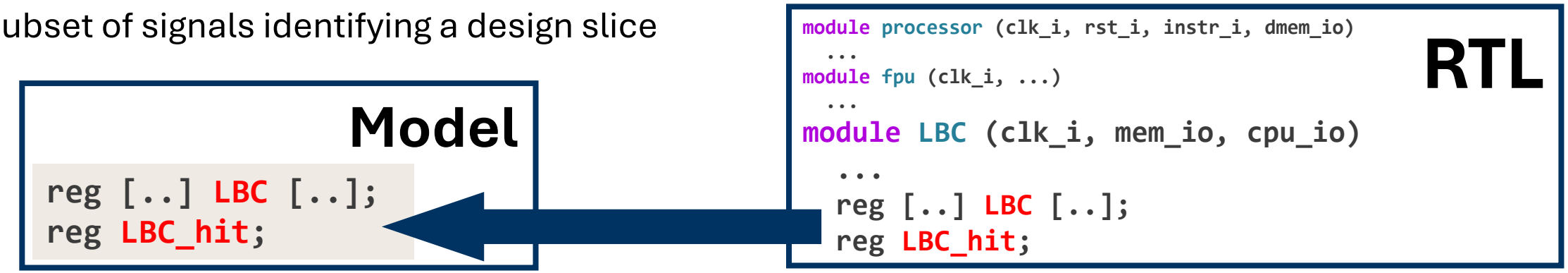
Abstract Microarchitectural Models: simpler than RTL, more detailed than ISA



The Micro-Update Model

Signals of interest (S)

Subset of signals identifying a design slice



The Micro-Update Model

Signals of interest (S)

Subset of signals identifying a design slice

Model

```
reg [..] LBC [..];
reg LBC_hit;
...
```

Micro-Updates

Imperatively defined operations (functions) on signals of interest

```
LBC_refill {
  LBC[index] <= mem_io.d;
  LBC_valid[index] <= 1;
}
```

```
LBC_serve (i) {
  cpu_io.d <= LBC[i];
  LBC_hit <= 1;
}
```

```
LBC_flush {
  LBC[0] <= 0;
  ...
  LBC[1] <= 0;
}
```

The Micro-Update Model

Signals of interest (S)

Subset of signals identifying a design slice

Micro-Updates

Imperatively defined operations (functions) on signals of interest

Guards and conditional execution

Boolean condition defining *when* a micro-update is triggered

```
guard_LBC_serve (i) :=
  is_ld(inst_mem) &&
  cpu.io.addr == LBC[i].addr
```

Model

```
reg [..] LBC [..];
reg LBC_hit;
...
```

```
LBC_refill () { ... }
LBC_serve (i) { ... }
LBC_flush ()  { ... }
...
```

The Micro-Update Model

Signals of interest (S)

Subset of signals identifying a design slice

Micro-Updates

Imperatively defined operations (functions) on signals of interest

Guards and conditional execution

Boolean condition defining *when* a micro-update is triggered

Model

```
reg [..] LBC [..];
reg LBC_hit;
...
```

```
LBC_refill () { ... }
LBC_serve (i) { ... }
LBC_flush ()  { ... }
...
```

```
guard_LBC_serve (i)
...
```

Formal Property: S-equivalence

Equivalence(RTL design D, model M, S):

cycle-wise functional behaviour of signals in S

in model M matches that of design D

S-equivalence *implies NI-soundness*

Equivalence(RTL design D, model M, S):

cycle-wise functional behaviour of signals in S

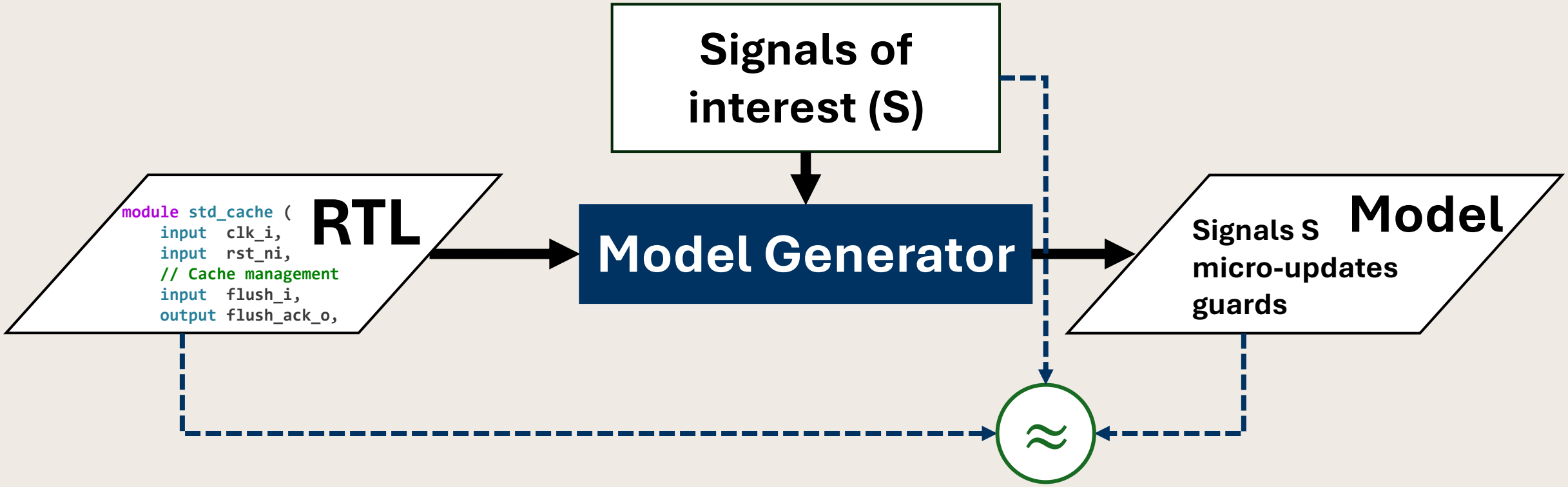
in model M matches that of design D

Soundness property:

For *any* non-interference property NI ***over S***,

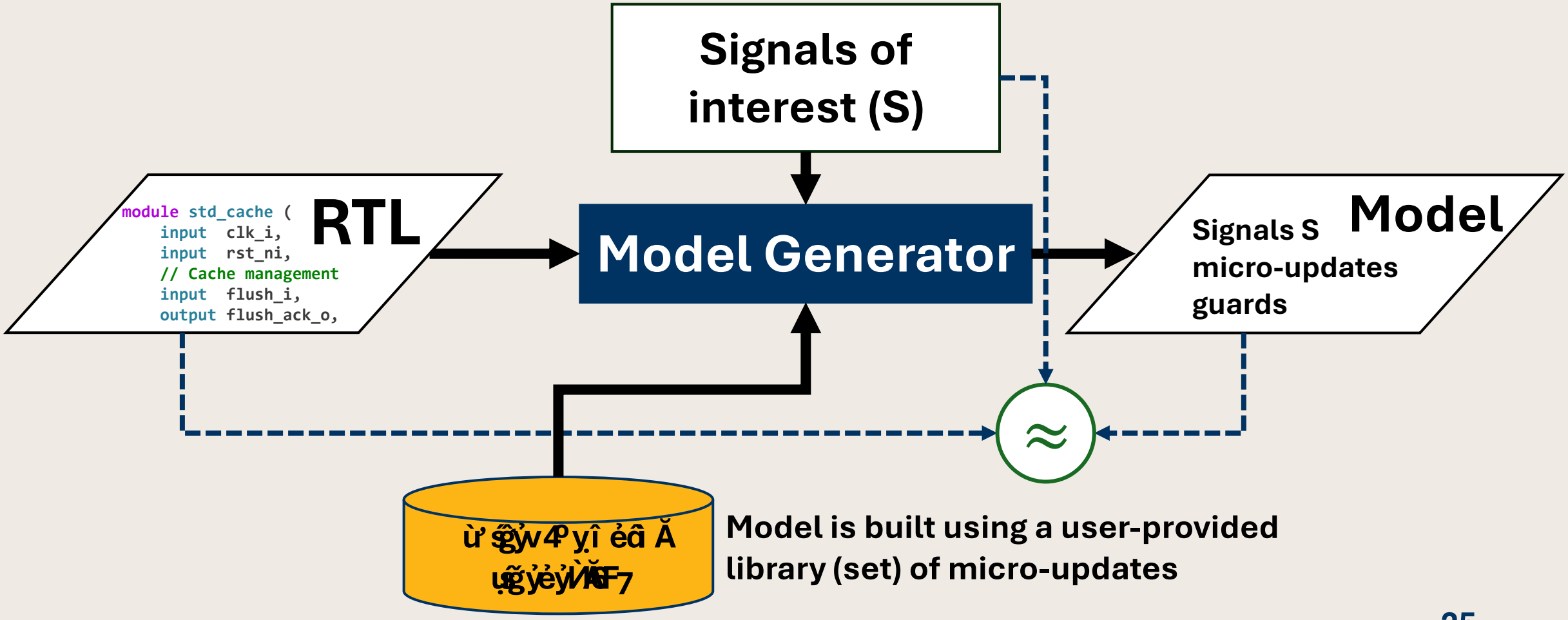
if model M satisfies NI then D also satisfies NI

Micro-update model synthesis problem



Model is S-equivalent to RTL design

Micro-update model synthesis problem

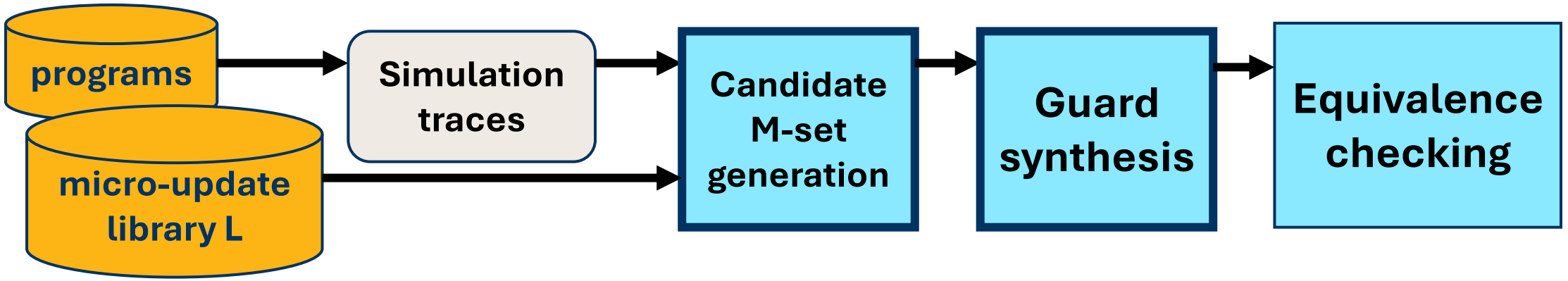


Model is built using a user-provided library (set) of micro-updates

Outline

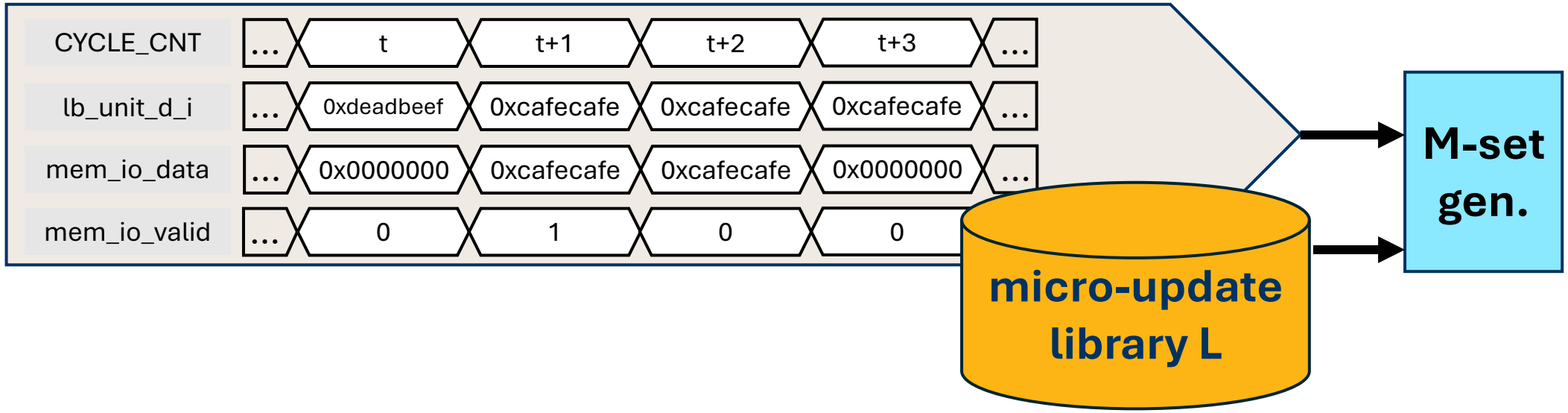
- Motivating Example
- Our formalism: the micro-update model
- Micro-update model synthesis problem
- **Synthesis Approach Highlights**
- Evaluation and Results

Synthesis Approach: High-level



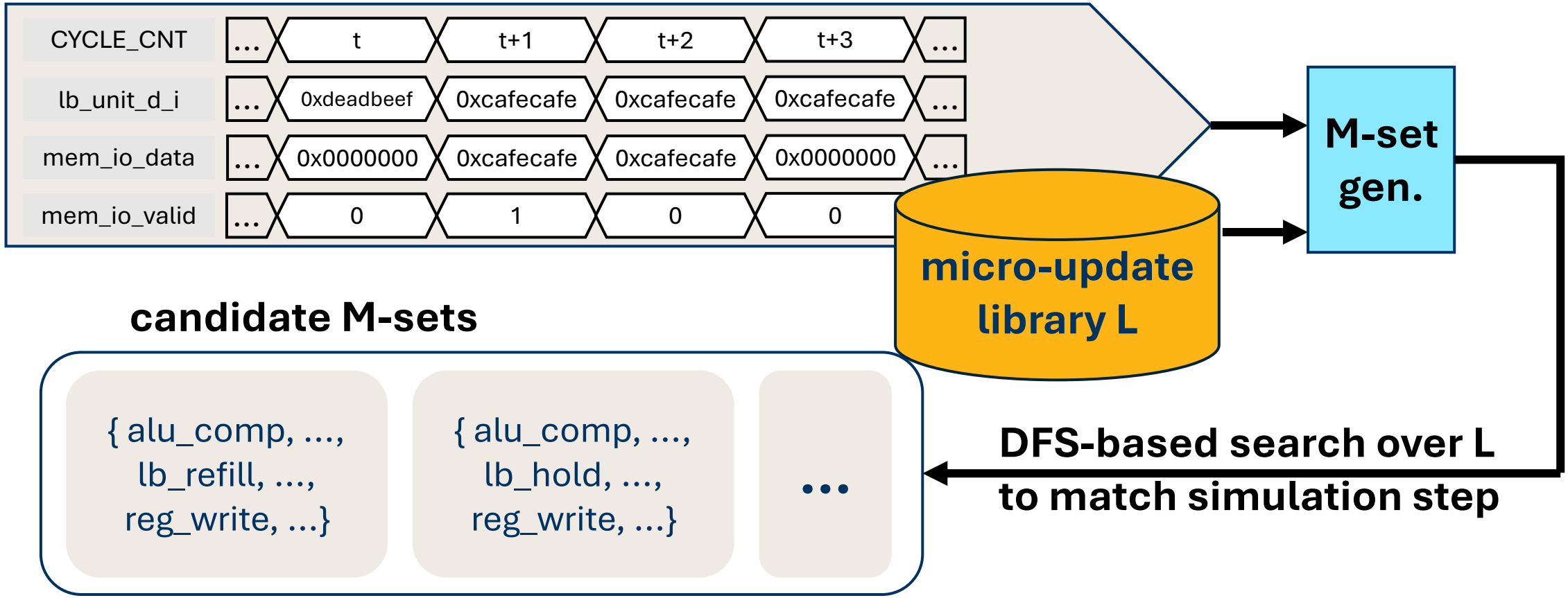
Synthesis Details: M-set generation

Generate viable sets of micro-updates for each simulation step



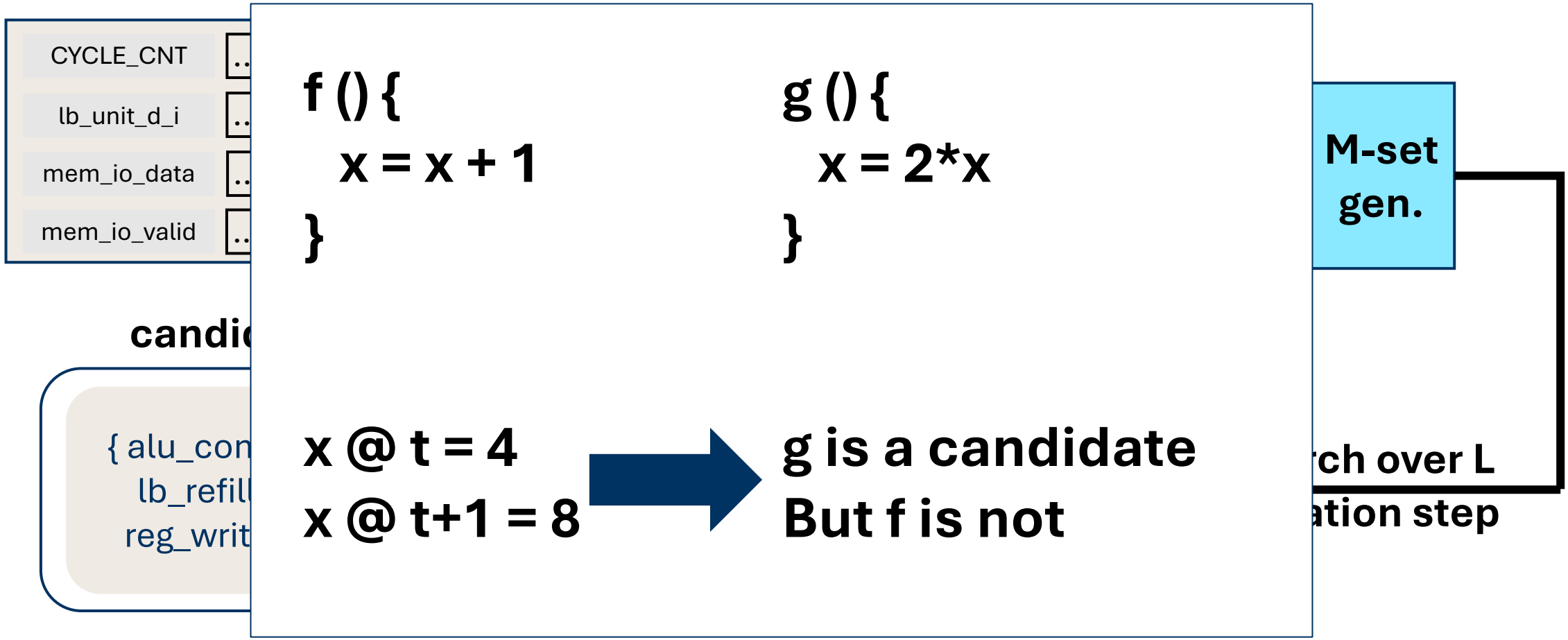
Synthesis Details: M-set generation

Generate viable sets of micro-updates for each simulation step



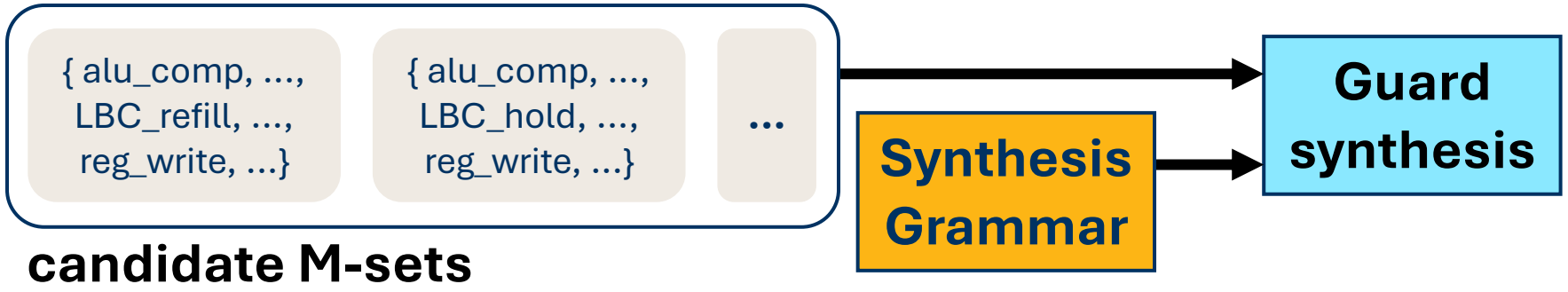
Synthesis Details: M-set generation

Generate viable sets of micro-updates for each simulation step



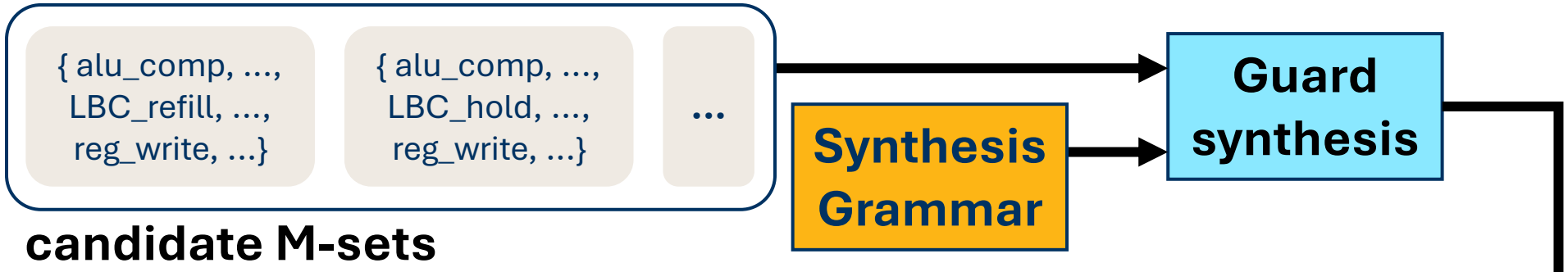
Synthesis Details: Guard Synthesis

Grammar-based search for guards consistent with candidates



Synthesis Details: Guard Synthesis

Grammar-based search for guards consistent with candidates




```
guard_LBC_refill =
      mem_io_valid &&
      is_load(inst_mem) && ...
```

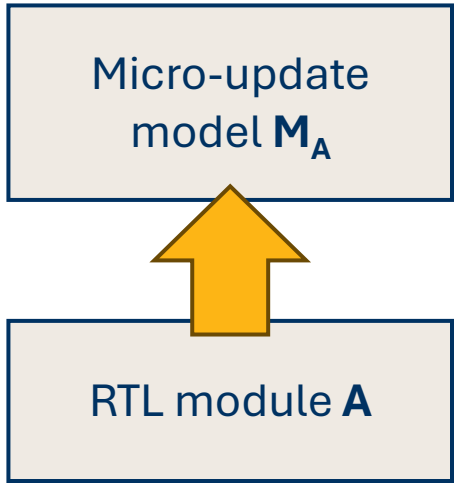
SyGuS encoding

"when a micro-update is not a candidate, its guard must evaluate to false"

SyGuS: Syntax Guided Synthesis

Hierarchical Synthesis

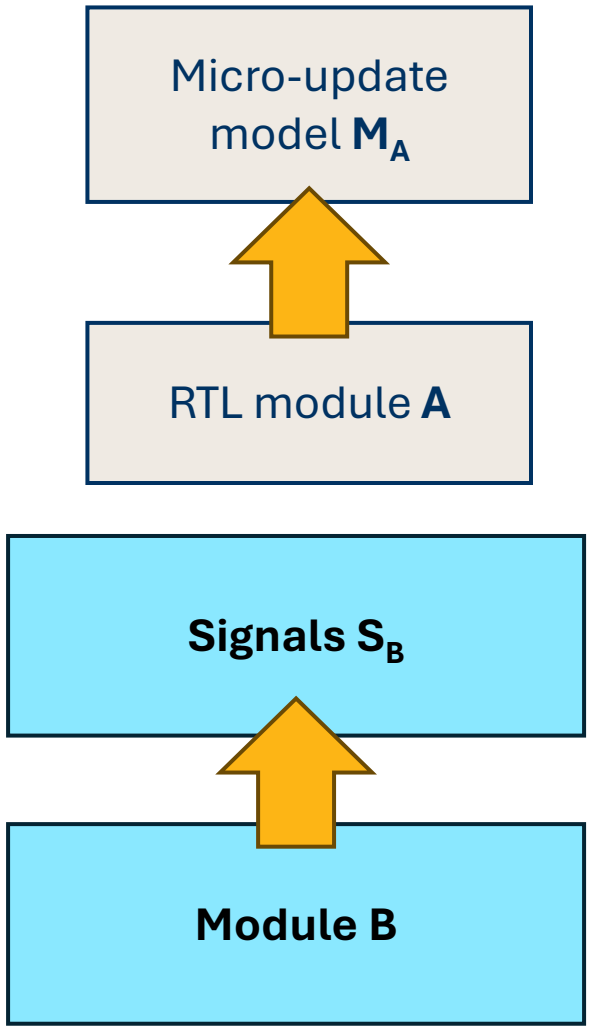
Model M_A for module A with signals S_A



Hierarchical Synthesis

Model M_A for module A with signals S_A

When synthesizing a model for a parent module B (with signals S_B)

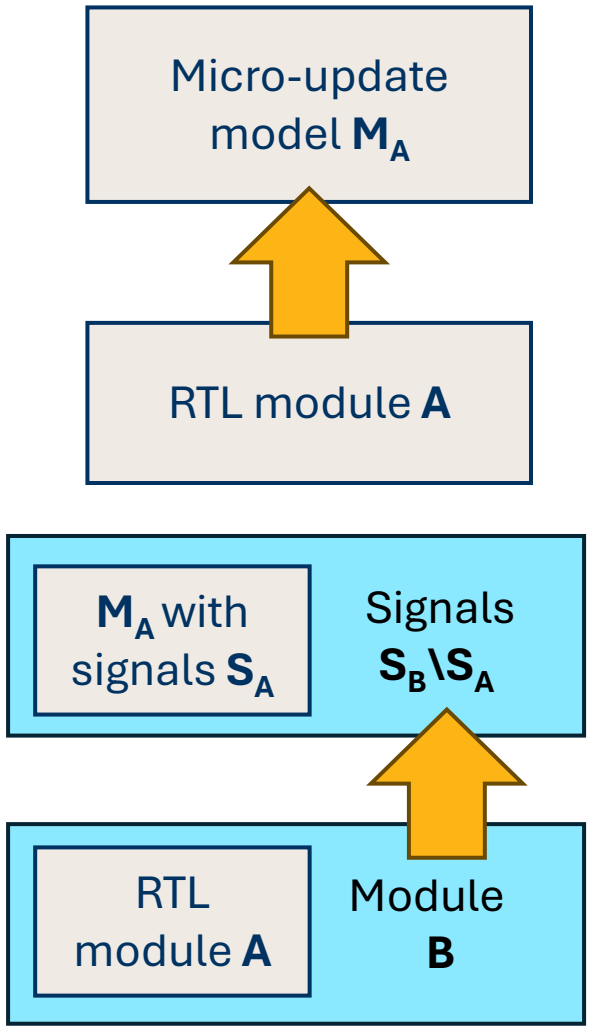


Hierarchical Synthesis

Model M_A for module A with signals S_A

When synthesizing a model for a parent module B (with signals S_B) **we can reuse M_A**

(Micro-update, Guard) pairs **compose** under certain “non-conflict” conditions



Outline

- Motivating Example
- Our formalism: the micro-update model
- Micro-update model synthesis problem
- Synthesis Approach Highlights
- **Evaluation and Results**

Outline

- Motivating Example
- Our formalism: the micro-update model
- Micro-update model synthesis problem
- Synthesis Approach Highlights
- **Evaluation and Results:**
  - **Evaluation of model lifting approach**
  - **Evaluation of security analysis using generated model**

Experimental Results: Synthesis

1. **Sodor processor family**

Experimental Results: Synthesis

1. **Sodor processor family**
2. **Ariane (cva6): 6-stage OoO-issue processor**
   - Generated models for memory subsystem components:
     wbuffer, TLB, load_store_unit
   - **Guard synthesis dominates runtime**

Model slice	Signals-of-interest	Guard synthesis	Equiv. proof
store_unit	S_1 = speculative/commit store queues, store request states	6m	3m
load_store_unit	$S_2 = S_1$ U load request states (valid/spec/commit/memresp.)	TO (> 2hour)	2m

Experimental Results: Synthesis

1. **Sodor processor family**
2. **Ariane (cva6): 6-stage OoO-issue processor**
   - Generated models for memory subsystem components:
     wbuffer, TLB, load_store_unit
   - **Guard synthesis dominates runtime**
   - **Hierarchical synthesis helps improve scalability**

Model slice	Signals-of-interest	Guard synthesis	Equiv. proof
store_unit	S_1 = speculative/commit store queues, store request states	6m	3m
load_store_unit	$S_2 = S_1$ U load request states (valid/spec/commit/memresp.)	TO (> 2hour) 11m	2m

Results: Security Verification

Non-interference-based security verification

Verification with generated model outperforms RTL design

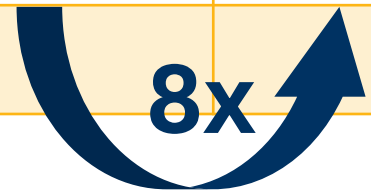
Symbolic testcase	Testcase constraint	Safe/ UnSafe	RTL Design runtime	Model runtime
alui sw lw alui	addr(**sw**) == addr(**lw**)	S	1m16s	**34s**
lw_1 sw lw_2 alui	addr(**sw**) == addr(**lw_2**)	S	1m48s	**44s**
lw_1 sw lw_2 lw_3	RE & rd(**lw_1**) != rs1(**sw**) & rd(**lw_1**) != rs1(**lw_2**)	US	9m4s	**1m2s**
lw_1 sw lw_2 lw_3	RE & rd(**lw_1**) != rs1(**sw**) & rd(**lw_1**) != rs1(**lw_3**)	S	12m36s	**1m29s**
	…			

Results: Security Verification

Non-interference-based security verification

Verification with generated model outperforms RTL design

Symbolic testcase	Testcase constraint	Safe/ UnSafe	RTL Design runtime	Model runtime
alui sw lw alui	addr(**sw**) == addr(**lw**)	S	1m16s	**34s**
lw_1 sw lw_2 alui	addr(**sw**) == addr(**lw_2**)	S	1m48s	**44s**
lw_1 sw lw_2 lw_3	RE & rd(**lw_1**) != rs1(**sw**) & rd(**lw_1**) != rs1(**lw_2**)	US	9m4s	**1m2s**
lw_1 sw lw_2 lw_3	RE & rd(**lw_1**) != rs1(**sw**) & rd(**lw_1**) != rs1(**lw_3**)	S	12m36s	**1m29s**
	…			



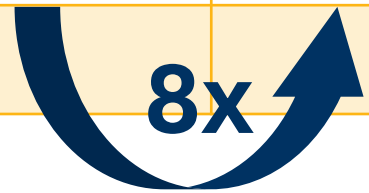
Results: Security Verification

Non-interference-based security verification

Verification with generated model outperforms RTL design

Symbolic testcase	Testcase constraint	Safe/ UnSafe	RTL Design	Model runtime
lw_1 sw lw_2 addi	addi(**sw**) == addi(lw_2)	S	11m48s	**44s**
lw_1 sw lw_2 lw_3	RE & rd(lw_1) != rs1(**sw**) & rd(lw_1) != rs1(lw_2)	US	9m4s	**1m2s**
lw_1 sw lw_2 lw_3	RE & rd(lw_1) != rs1(**sw**) & rd(lw_1) != rs1(lw_3)	S	12m36s	**1m29s**
	...			

S-equivalence ==> NI-soundness



Conclusion

- **Micro-update models**: operationally represent cycle-accurate functional behaviour of a design slice
- **Lifting algorithm:** base on M-set generation and guard synthesis.

  Hierarchical synthesis improves scalability!
- **Evaluation:** We evaluate the synthesis algorithm and the advantage of performing verification with generated models

  8x (and growing) verification performance improvement over source RTL!

END